2021华为软件精英挑战赛 上合赛区32强二等奖

题目大概意思是我们是公有云提供商，有很多类型服务器（不同的core和memory，价格，每天能耗）可以购买，给用户提供不同类型的虚拟机服务。而且虚拟机还分单双节点。我们如何在用户虚拟机请求给定的情况下，购买充足服务器满足用户所有需求，但是成本最低。这个成本包含服务器购买成本和服务器运行能耗成本。并且每天还有一定次数可以将部署好的虚拟机迁移到其他服务器上去，以便能将服务器尽可能满，空出一些服务器，减少能耗成本。所以本题的关键点在于服务器的购买，虚拟机在服务器上的部署，虚拟机的迁移。

**服务器的选择：**

我们最开始就是在满足最大虚拟机core和memory的最低需求的情况下用服务器的能耗来进行服务器的购买。但是我当时想到服务器用的天数多的时候，每天能耗的价格占主导因素，而服务器用的天数少的时候，服务器的购买价格占主导因素。然后我对数据进行了分析，发现购买价格是能耗的250倍，然后我就用250做了一个判断，小于250天时用购买价格做判断，选价格低的。我发现成本少了几千万。证明是有效果的。 但是参数越少越好，最好弄成自适应的，因为线上数据集的分布其实不清楚的。在core和memory相同的情况下，serverCost/powerCost确定了服务器的购买价格是能耗价格的倍数，则剩余天数小于这个倍数的时候应该使用服务器的价格购买服务器。

我们又对服务器数据进行了分析，发现服务器的购买价格以及每天的能耗都是core和memory的二元函数，即购买价格和core、memory基本处于一个平面上，所以我们进行了最小二乘拟合，得到表达式a1\*core+b1\*memory=serverCost, a2\*core+b2\*memory=powerCost。。所以我们用拟合得到的表达式 T\_days-(a1+b1)/(a2+b2)作为分界线，之前的用powerCost，之后的用serverCost。

在这个比赛中，服务器的选择是比较重要的。我们将购买的服务器根据core/memory的比值分成多类（初赛中分成了三类，复赛中分成了两类）。然后根据请求的虚拟机数据分布得到较优的分隔核存比，具体方式是将请求中Add的虚拟机根据核存比排序，然后取到的用于分隔的核存比要保证分隔之后的请求的需求资源差不多相等。这样做的好处是使得各类服务器能均衡的得到请求，不至于造成某类服务器请求量非常大，某类非常小。首先我们采用的购买策略是在保证最大需求的情况下，尽可能的买小的服务器，这个策略在初赛阶段表现得还行，但是在复赛阶段这种方式达到一个瓶颈，原因在于买的服务器太小容易后面造成很多资源碎片，造成资源浪费。而且服务器买得小意味着需要购买更多的服务器，对迁移方面的时间复杂度造成很大的负担。因此之后我们对服务器数据进行了分析，发现服务器的购买价格以及每天的能耗都是core和memory的二元函数，即购买价格和core、memory基本处于一个平面上，所以我们进行了最小二乘拟合，得到表达式a1\*core+b1\*memory=serverCost, a2\*core+b2\*memory=powerCost。而我们讨论中想到当服务器用的天数多的时候，每天能耗的价格占主导因素，而服务器用的天数少的时候，服务器的购买价格占主导因素。我们开始设定了一个剩余天数的阈值，当小于250天的时候，利用serverCost选服务器，否则用powerCost选阈值，发现少了几千万，证明是有效果的。但是参数越少越好，最好弄成自适应的。所以我们用拟合得到的表达式T\_days-(a1+b1)/(a2+b2)作为分界线，之前的用powerCost，之后的用serverCost。此外，我们还尝试过其他购买服务器的策略，

**虚拟机的分配：**

在初赛阶段我们将服务器的购买和虚拟机的分配都分成了单双节点类型，即单节点虚拟机只分配在单节点服务器池，双节点虚拟机只分配在双节点服务器池。这样设计的好处是不会造成很大的负载失衡。在复赛阶段我们达到一个瓶颈之后，取消了单双节点的分开机制，单双节点虚拟机在一个服务器资源池中请求资源，达到了不错的效果，这样做的好处是**减小请求粒度，进而减少资源碎片**。其缺点是可能会带来负载失衡的情况，这种情况我们有考虑在匹配的时候或者在迁移的时候避免或者修复负载失衡的状态。但是带来的效果甚微，这与数据集的分布有非常大的关系。

对于虚拟机的分配问题，其实是一种bin-packing（装箱）问题，这是一种NP-hard问题，无法在多项式时间内求得最优解。起初我们利用局部贪心的思想，将装箱问题简化为背包问题，即对于某一个服务器来说，其装配是最优的。每个服务器可以看成一个背包，最大容量就是最大的core和memory，如果能够将每个背包尽可能装满，则购买的服务器会较少，能耗也会较低，因为就可以有空闲的服务器不增加能耗。这个背包问题是一个二维费用背包问题，这里有两个容量限制，即core和memory，目标是尽可能的装满服务器，因此可用占用的core+memory的值来衡量其价值。较01背包来说，多了一个费用的维度，因此可用三维的动态规划解决。状态设计为dp[i][v][u]：表示只使用前i个虚拟机，当前core容量还剩v，memory容量还剩u时的最大价值（core+memory）。状态转移方程为：

此外，需要记录过程中状态转移路径，因此定义了一个三维数组path[i][v][u]: 表示当前状态下取最优值时的取最后一个物品（虚拟机）的位置。其转移方程为：

为了提高运行效率，我们利用二维滚动数组的方式将f[i][v][u]优化为f[v][u]，实际上仍然是一个三维动态规划问题，只是将空间复杂度优化为二维。

但是利用背包进行匹配的效果不太好，在该问题下使用背包问题是一种局部最优的思想，只能保证当前服务器的装配是最优的，针对整体来说并不是最优的，而且由于请求中夹杂着删除虚拟机的操作，很多时候之前装配好的服务器到后面由于删除一些虚拟机之后又空出一部分空间。所以利用背包问题来解决这种动态的问题效果并不佳。并且这种方法还很耗时，所以最终我们摒弃了这种算法。

转而使用比较朴素的first fit decreasing，best fit这类方法。最开始我们使用best fit算法耗时太长，但是后面我们更改了服务器的存储结构，将其用set存储，自定义排序方式按照资源剩余率排序，剩的少的在前面，则虚拟机部署时从前往后找到的第一个服务器就是剩余容量差不多能装上的。不过set其实设计初衷就不能直接修改其中的值，只有删除和插入节点才会触发重排，并且只重排插入删除节点相关的节点，这样才能达到logn的时间复杂度。最终我们对于需要修改的服务器先将存在set中的指针删除，然后更改容量，然后再插入。

**虚拟机的迁移：**

我主要负责迁移部分代码的编写。迁移的目的就是将每个服务器尽可能装满，然后空出一些服务器，这样就可以减少能耗成本。本场景中迁移有数量限制，即每天的迁移量不得超过虚拟机数量的3%。最开始我们是将所有的服务器按照部署的虚拟机的数量排序（也尝试过根据服务器上负载的虚拟机数量升序、服务器的能耗升序），多的在前，然后将后面服务器上的虚拟机往前面迁移。但是因为O(nm)的复杂度(n为服务器的数量，m为虚拟机的数量)，所以速度比较慢。后来我们在迁移之前将剩余率<0.04的服务器排除，这样虽然还是O(nm)的复杂度，但是时间常数可以降低不少。后面我又发现可以将当天需要删除的虚拟机先迁移到后面，将当天不删除的迁移到前面，这样每天之后删除的vm部署的服务器就很可能空出来，这样又提升了一些分数。

最终我们初赛22名，复赛12名，夺得上合赛区32强二等奖。